UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/748,098	12/21/2000	Nicholas J. Kelsey	20880-05093	4360

758 7590 05/23/2007

FENWICK & WEST LLP
SILICON VALLEY CENTER
801 CALIFORNIA STREET
MOUNTAIN VIEW, CA 94041

EXAMINER	
HUISMAN, DAVID J	

ART UNIT	PAPER NUMBER
2183	

MAIL DATE	DELIVERY MODE
05/23/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	09/748,098	KELSEY ET AL.	
	Examiner	Art Unit	
	David J. Huisman	2183	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *27 March 2007*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-55* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-55* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☒ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *21 December 2000* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date *2/3/2007*.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. Claims 1-55 have been examined.

Papers Submitted

2. It is hereby acknowledged that the following papers have been received and placed of record in the file: IDS as received on 2/3/2007 and Amendment as received on 3/27/2007.

Specification

3. The disclosure is objected to because of the following informalities: On page 23, line 17, replace "thread, although" with --thread. Although--. Appropriate correction is required.

Claim Objections

4. Claim 53 is objected to because of the following informalities: In line 2, replace "a" with --an--. Appropriate correction is required.

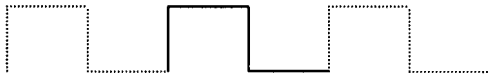
Claim Rejections - 35 USC § 112

5. The following is a quotation of the first paragraph of 35 U.S.C. 112:

> The specification shall contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, or with which it is most nearly connected, to make and use the same and shall set forth the best mode contemplated by the inventor of carrying out his invention.

6. Claims 2-18 are rejected under 35 U.S.C. 112, first paragraph, as failing to comply with the enablement requirement. The claim(s) contains subject matter which was not described in the specification in such a way as to enable one skilled in the art to which it pertains, or with

which it is most nearly connected, to make and/or use the invention. More specifically, regarding claim 17, the examiner does not understand how one could make or use an invention that switches threads **between consecutive instruction cycles**. As is known, an instruction cycle (or clock cycle) is a period of time in which a clock oscillates from low to high. The cycle then repeats. So, for instance, as shown below, a first cycle (dotted) is followed by a second cycle (solid), which is followed by a third cycle (dotted).



At any given time, the system is in one of the cycles and it is not clear at what point the system is in between cycles. For instance, the leftmost edge in the above figure is the rising edge of the first clock cycle. The rising edge is part of the first cycle, as is everything else up until the rising edge of the second cycle (the first solid edge). There is no between-cycle period because the system is always in a cycle. The examiner believes that applicant is trying to claim that thread switching occurs without incurring a time penalty. However, if this is the case, then the claim should include this claim language instead of claiming "between instruction cycles," which the examiner feels is not enabled and impossible. It simply appears, from page 17, lines 9-13, of the specification, that applicant is equating switching with zero time-penalty with switching "between instructions" (or instruction cycles, as claimed). Or, more specifically, applicant appears to be saying that because switching takes place with no time penalty, the system appears to be or is effectively switching between cycles. However, the examiner again challenges how

the system can physically switch between cycles when there is no time in between cycles for switching. Since it is unknown when the system is between cycles, it is also unknown how to make a system which switches threads between cycles. Claims 2-16 and 18 are not enabled because they are dependent on a claim that is not enabled. Applicant is asked to clarify and make appropriate corrections.

7. Claims 19-28 are rejected under 35 U.S.C. 112, first paragraph, as failing to comply with the enablement requirement. The claim(s) contains subject matter which was not described in the specification in such a way as to enable one skilled in the art to which it pertains, or with which it is most nearly connected, to make and/or use the invention. More specifically, regarding claim 19, the examiner does not understand how one could make or use an invention that switches threads **between the end of an execution cycle and before the beginning of a next consecutive execution cycle**. As is known, an execution cycle (or clock cycle) is a period of time in which a clock oscillates from low to high. The cycle then repeats. So, for instance, as shown below, a first cycle (dotted) is followed by a second cycle (solid), which is followed by a third cycle (dotted).

At any given time, the system is in one of the cycles and it is not clear at what point the system is in between cycles. For instance, the leftmost edge in the above figure is the rising edge of the first clock cycle. The rising edge is part of the first cycle, as is everything else up until the rising edge of the second cycle (the first solid edge). There is no between-cycle period because

the system is always in a cycle. The examiner believes that applicant is trying to claim that thread switching occurs without incurring a time penalty. However, if this is the case, then the claim should include this claim language instead of claiming "between instruction cycles," which the examiner feels is not enabled and impossible. It simply appears, from page 17, lines 9-13, of the specification, that applicant is equating switching with zero time-penalty with switching "between instructions" (or instruction cycles, as claimed). Or, more specifically, applicant appears to be saying that because switching takes place with no time penalty, the system appears to be or is effectively switching between cycles. However, the examiner again challenges how the system can physically switch between cycles when there is no time in between cycles for switching. Since it is unknown when the system is between cycles, it is also unknown how to make a system which switches threads between cycles. Claims 20-28 are not enabled because they are dependent on a claim that is not enabled. Applicant is asked to clarify and make appropriate corrections.

8. The following is a quotation of the second paragraph of 35 U.S.C. 112:

> The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

9. Claims 2-29 are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

10. Claim 17 is rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention. Specifically, switching from the first thread state to the second thread state

"between consecutive instruction cycles" is indefinite because it is unclear what applicant means by "between consecutive instruction cycles". As explained above, the system, at any point in time, is operating during an instruction cycle and therefore, it is not known when the system is between cycles. For purposes of examination, the examiner will interpret this to mean that the switching occurs without incurring a time penalty. The two are apparently equivalent according to the specification (page 17, lines 9-12).

11. Claim 19 is rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention. Specifically, switching from the first thread state to the second thread state "between the end of an execution cycle and before the beginning of a next consecutive execution cycle" is indefinite because it is unclear what applicant means by "between". As explained above, the system, at any point in time, is operating during an execution cycle and therefore, it is not known when the system is between cycles. For purposes of examination, the examiner will interpret this to mean that the switching occurs without incurring a time penalty. The two are apparently equivalent according to the specification (page 17, lines 9-12).

12. Claims 2-16, 18, and 20-29 are rejected under 35 U.S.C. 112, 2nd paragraph, as being indefinite, because they are dependent, either directly or indirectly, on an indefinite claim.

Claim Rejections - 35 USC § 102

13. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

14. Claims 1, 29-33, 42, 43, and 45-47 are rejected under 35 U.S.C. 102(e) as being anticipated by Borkenhagen et al., U.S. Patent No. 6,076,157 (herein referred to as Borkenhagen).

15. Referring to claim 1, Borkenhagen has taught a computer based system for switching between program contexts comprising:

a) a processor capable of having a first program thread and a second program thread in an execution pipeline having a thread selection hardware. See Fig.4A and note that the processor includes thread switch (selection) hardware which selects between a first thread (thread 0) and a second thread (thread 1). Also, see column 7, lines 15-20, and note that the processor has an execution pipeline.

b) a first set of data storage devices capable of storing a first thread state of said processor. See Fig.4A, component 442, and column 10, lines 18-56. Note that there is a first set of storage devices for storing a group of bits which represent the state of the first thread.

c) a second set of data storage devices capable of storing a second thread state of said processor. See Fig.4A, component 444, and column 10, lines 18-56. Note that there is a second set of storage devices for storing a group of bits which represent the state of the second thread.

d) a hardware thread scheduler for identifying which of said program threads said processor executes and configurable to allocate available processing time of the processor among at least the first and second program threads by causing thread-switching at a fixed time according to a predetermined fixed schedule, said schedule specifying that the first thread should be allocated

processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles. See column 13, line 20, to column 14, line 22, column 14, line 63, to column 15, line 3, and claim 5 of Borkenhagen. In these passages it is taught that threads are scheduled with the help of a thread switch control register which includes bits that each correspond to a different thread switch event. It is disclosed that any of the bits may be enabled/disabled to achieve switch flexibility for the associated thread. One of the bits is bit 9, which calls for switching when the thread switch time-out value is reached. Given two threads and given that bit 9 is enabled, a first thread will execute for X cycles before being switched, and then a second thread will execute for Y cycles before being switched, where X and Y are different. Then the process will repeat since there are only two threads in one embodiment (column 5, lines 7-14). For simplicity, assume that the first thread (thread A) has a timeout value of 4 cycles while the second thread (thread B) has a timeout value of 2 cycles. The execution of these threads would be as follows:

A A A A B B A A A A B B A A A A B B

Note that thread A, when it is not executing, is allocated time every 2 cycles while thread B, when it is not running, is allocated time every 4 cycles. That is, when thread B is executing, after 2 cycles, thread A will be allocated 4 cycles of time. Likewise, when thread A is executing, after 4 cycles, thread B will be allocated 2 cycles of time.

16. Referring to claim 29, Borkenhagen has taught a system as described in claim 1. Borkenhagen has further taught that said thread selection hardware in the pipelined processor switches between said first and second thread state after the end of the execution of a first program instruction in the first thread and before the beginning of the execution of a second program instruction. This is deemed inherent because thread A will execute for some amount of time and then a switch will occur to another thread. The switching marks the end of executing an instruction from thread A and the beginning of executing an instruction in thread B. And, clearly, the system must be in the second state before it can begin executing the second thread.

17. Referring to claim 30, Borkenhagen has taught a system as described in claim 1. Borkenhagen has further taught that said processor is an embedded pipelined processor. See claim 14 of Borkenhagen. The processor, which is pipelined, is embedded in a system.

18. Referring to claim 31, Borkenhagen has taught a system as described in claim 1. Borkenhagen has further taught that said first state is the state of the processor during the execution of the first program thread. See column 10, lines 18-56.

19. Referring to claim 32, Borkenhagen has taught a system as described in claim 1. Borkenhagen has further taught that said second state is the state of the processor during the execution of the second program thread. See column 10, lines 18-56.

20. Referring to claim 33, Borkenhagen has taught a system as described in claim 1. Borkenhagen has further taught that said processor switches between said first and second state by changing a state selection register. See column 15, lines 1-7. The decrement register (state selection register) is decremented each cycle until it gets to zero and then a thread switch occurs.

Therefore, the processor switches between first and second state by changing a state selection register.

21. Referring to claim 42, Borkenhagen has taught a system as described in claim 1. Borkenhagen has further taught that said processor is capable of restoring said second state of said processor during execution of said first program thread. See Fig.4A, component 450, and note that at some point during execution of the first thread, a thread switch will occur, and the second state is restored. That is, the switching to (restoring the state of) the second thread occurs during execution of the first thread.

22. Referring to claim 43, Borkenhagen has taught a system as described in claim 1. Borkenhagen has further taught that said processor is capable of storing said second thread state of said processor during execution of said first program thread. See column 13, lines 20-45. This control register (and control state) allows the system to specify which types of events would result in the switching of the associated thread, thereby increasing flexibility by allowing the user to choose how the thread may or may not be switched. This register may be set by any thread for itself or another thread.

23. Referring to claim 45, Borkenhagen has taught a system as described in claim 1. Borkenhagen has further taught that the predetermined fixed schedule is one of a fixed strict schedule, a semi-flexible strict schedule, and a loose strict schedule. Borkenhagen has taught at least a fixed strict schedule where threads are switched based on timeout values. For instance, see claim 5 of Borkenhagen.

24. Referring to claim 46, Borkenhagen has taught a computer based method for switching between program contexts in a multithreading pipelined processor (see Fig.4A and the abstract)

having a hardware thread selector (see Fig.4A) and an execution pipeline (see column 7, lines 15-20 and note that the processor has an execution pipeline), the method comprising:

a) storing a first context of said processor in a first set of data storage devices comprising a first thread state corresponding to a first program thread. See Fig.4A, component 442, and column 10, lines 18-56. Note that there is a first set of storage devices for storing a group of bits which represent the state of the first thread.

b) storing a second context of said processor in a second set of data storage devices comprising a second thread state corresponding to a second program thread. See Fig.4A, component 444, and column 10, lines 18-56. Note that there is a second set of storage devices for storing a group of bits which represent the state of the second thread.

c) switching the processor from the first thread state to the second thread state by coupling the execution pipeline from the first set of data storage devices to the second set of storage devices via the hardware thread selector at a fixed time according to a predetermined fixed execution schedule, said execution schedule specifying that the processor should switch to the first thread state every first number of cycles and that the processor should switch to the second thread state every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles. See column 13, line 20, to column 14, line 22, column 14, line 63, to column 15, line 3, and claim 5 of Borkenhagen. In these passages it is taught that threads are scheduled with the help of a thread switch control register which includes bits that each correspond to a different thread switch event. It is disclosed that any of the bits may be enabled/disabled to achieve switch flexibility for the associated thread. One of the bits is bit 9, which calls for switching when the thread switch time-out value is reached. Given two threads and given that

bit 9 is enabled, a first thread will execute for X cycles (using its respective state storage - Fig.4A, component 442) before being switched, and then a second thread will execute for Y cycles (using its respective state storage - Fig.4A, component 442) before being switched, where X and Y are different. Then the process will repeat since there are only two threads in one embodiment (column 5, lines 7-14). For simplicity, assume that the first thread (thread A) has a timeout value of 4 cycles while the second thread (thread B) has a timeout value of 2 cycles. The execution of these threads would be as follows:

A A A A B B A A A A B B A A A A B B

Note that thread A, when it is not executing, is allocated time every 2 cycles while thread B, when it is not running, is allocated time every 4 cycles. That is, when thread B is executing, after 2 cycles, thread A will be allocated 4 cycles of time. Likewise, when thread A is executing, after 4 cycles, thread B will be allocated 2 cycles of time.

25. Referring to claim 47, Borkenhagen has taught a method as described in claim 46. Borkenhagen has further taught that the switching comprises changing a state selection register included in the hardware thread selector. See column 15, lines 1-7. The decrement register (state selection register), which is part of the thread selector, is decremented each cycle until it gets to zero and then a thread switch occurs. Therefore, the processor switches between first and second states by changing a state selection register.

Claim Rejections - 35 USC § 103

26. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

27. Claims 2-4, 13, 16-17, and 19-24 are rejected under 35 U.S.C. 103(a) as being unpatentable over Joy et al., U.S. Patent No. 6,542,991 (as applied in the previous Office Action and herein referred to as Joy), in view of Emer et al., U.S. Patent No. 6,493,741 (as applied in the previous Office Action and herein referred to as Emer).

28. Referring to claim 17, Joy has taught a computer based system for switching between program contexts comprising:

a) a pipelined processor (Fig.3, component 300; column 8, lines 14-67) capable of having a first program thread and a second program thread in an execution pipeline having a thread selection hardware (see Fig.6 and note the "thread select logic"; column 13 lines 5-23, column 15 lines 4-7), the execution pipeline including a set of stages for executing instructions and configured to execute a single instruction at each different stage in the set of stages (see claim 1, for instance, and note the existence of a pipeline). It should be noted that pipelines inherently include stages which perform different tasks, and in which a different instruction may be executing at one time.

b) a first set of data storage devices capable of storing a first thread state of said pipelined processor. See Fig.3, component 310, and column 8, lines 27-44. Note that component 310 includes flip-flops and a register file structure for storing a first thread state (for thread 0).

c) a second set of data storage devices capable of storing a second thread state of said pipelined processor. See Fig.3, component 312, and column 8, lines 27-44. Note that component 312 includes flip-flops and a register file structure for storing a second thread state (for thread 1).

d) a hardware thread scheduler for identifying which of said program threads said pipelined processor executes (Fig.6; column 15, lines 4-7) and configurable to allocate available processing time of the pipelined processor among at least the first and second program threads according to an execution schedule. See column 3, lines 33-51, and note that Joy's thread-switching may be of the oblivious type, in which threads are switched every N cycles without notification of stalling.

e) Joy has not explicitly taught that said thread selection hardware in the pipelined processor switches from said first thread state to said second thread state between consecutive instruction cycles in response to the hardware thread scheduler identifying which of said program threads said pipelined processor executes. However, recall that Joy has taught that threads may be switched every N cycles. Since there is no disclosed restriction as to what value N might be, a thread switch may occur every cycle (N=1). When N=1, a fine-grained multithreaded system is achieved, as is known in the art. Emer has taught such a fine-grained system in which threads are switched every cycle (in consecutive cycles). See Fig.1(b) and column 1, lines 52-65. Such a system eliminates vertical waste, thereby increasing throughput. Furthermore, in some instances, stalls for a particular thread are at least partially, and sometimes fully, masked. That is, if a thread may stall for a maximum period of X cycles, then if the system includes X threads, the stall will never affect the system. For example, assume that a system has 4 threads, and a thread may stall for a maximum of 4 cycles. In cycle 0, thread 0 must stall, in cycle 1, thread 1

will execute, in cycle 2, thread 2 will execute, in cycle 3, thread 3 will execute, and then in cycle 4, by the time thread 0 is to execute again, the stall time would have elapsed. The stall is effectively masked due to the switching every cycle. As a result, in order to eliminate vertical waste, increase throughput, and mask stalling in some instances, since Joy discloses that threads may be switched every N cycles, it would have been obvious to one of ordinary skill in the art at the time of the invention to have N=1 and have Joy switch threads on consecutive cycles, as taught by Emer.

29. Referring to claim 2, Joy in view of Emer has taught a system as described in claim 17. Joy has further taught that said first thread state is the thread state of the processor during the execution of the first program thread. See Fig.3, component 310, and column 8, lines 27-44. Note that component 310 includes flip-flops and a register file structure for storing a first thread state (for thread 0).

30. Referring to claims 3, Joy in view of Emer has taught a system as described in claim 17. Joy has further taught that said second thread state is the thread state of the processor during the execution of the second program thread. See Fig.3, component 312, and column 8, lines 27-44. Note that component 312 includes flip-flops and a register file structure for storing a second thread state (for thread 1).

31. Referring to claim 4, Joy in view of Emer has taught a system as described in claim 17. Joy has further taught that said processor switches between said first and second thread state by changing a state selection register. See Fig.5 and column 13, lines 5-64. The thread select logic includes a flip-flop for each thread, where the flip-flops collectively form a register that includes an active bit in the position corresponding to the active thread.

32. Referring to claim 13, Joy in view of Emer has taught a system as described in claim 17. Joy has further taught that said processor is capable of restoring said second thread state of said processor during execution of said first program thread. See column 6, lines 15-35. Clearly, a thread is executed until a switch signal is given. Therefore, switching to (restoring a) thread occurs during execution of another thread.

33. Referring to claim 16, Joy in view of Emer has taught a system as described in claim 17. Joy has further taught that the fixed schedule is one of a fixed strict schedule, a semi-flexible strict schedule, and a loose strict schedule. From column 3, lines 28-51, switching every N cycles is considered a fixed strict schedule, i.e., a thread switch must occur every N cycles.

34. Referring to claim 19, Joy has taught a computer based method for switching between program contexts in a multithreading pipelined processor (Fig.3; column 8, lines 14-67) having a hardware thread selector (Fig.6) and an execution pipeline (see claim 1 of Joy, for instance), the execution pipeline including a set of stages for executing instructions and configured to execute a single instruction at each different stage of the set of stages (it should be noted that pipelines inherently include stages which perform different tasks, and in which a different instruction may be executing at one time), the method comprising:

a) storing a first context of said pipelined processor in a first set of data storage devices, the first context corresponding to a first program thread. See Fig.3, component 310, and column 8, lines 27-44. Note that component 310 includes flip-flops and a register file structure for storing a first thread state (for thread 0).

b) storing a second context of said pipelined processor in a second set of data storage devices, the second context corresponding to a second program thread. See Fig.3, component 312, and

column 8, lines 27-44. Note that component 312 includes flip-flops and a register file structure for storing a second thread state (for thread 1).

c) Joy has not explicitly taught switching the pipelined processor from executing the first program thread to executing the second program thread between the end of an execution cycle and before the beginning of a next consecutive execution cycle by coupling the execution pipeline from the first set of data storage devices to the second set of storage devices via the hardware thread selector. However, recall that Joy has taught that threads may be switched every N cycles (column 3, lines 33-38). Since there is no disclosed restriction as to what value N might be, a thread switch may occur every cycle (N=1). When N=1, a fine-grained multithreaded system is achieved, as is known in the art. Emer has taught such a fine-grained system in which threads are switched every cycle (in consecutive cycles). See Fig.1(b) and column 1, lines 52-65. Such a system eliminates vertical waste, thereby increasing throughput. Furthermore, in some instances, stalls for a particular thread are at least partially, and sometimes fully, masked. That is, if a thread may stall for a maximum period of X cycles, then if the system includes X threads, the stall will never affect the system. For example, assume that a system has 4 threads, and a thread may stall for a maximum of 4 cycles. In cycle 0, thread 0 must stall, in cycle 1, thread 1 will execute, in cycle 2, thread 2 will execute, in cycle 3, thread 3 will execute, and then in cycle 4, by the time thread 0 is to execute again, the stall time would have elapsed. The stall is effectively masked due to the switching every cycle. As a result, in order to eliminate vertical waste, increase throughput, and mask stalling in some instances, since Joy discloses that threads may be switched every N cycles, it would have been obvious to one of

ordinary skill in the art at the time of the invention to have N=1 and have Joy switch threads on consecutive cycles, as taught by Emer.

35. Referring to claim 20, Joy in view of Emer has taught a system as described in claim 19. Joy has further taught that the switching comprises changing a state selection register included in the hardware thread selector. See Fig.5 and column 13, lines 5-64. The thread selector logic includes a flip-flop for each thread, where the flip-flops collectively form a register that includes an active bit in the position corresponding to the active thread. When a thread is switched, then a new bit in the register would be set while the previously set bit is reset.

36. Referring to claim 21, Joy in view of Emer has taught a method as described in claim 19. Joy has further taught identifying which of the said program threads said processor executes according to an execution schedule. See column 3, lines 33-51.

37. Referring to claim 22, Joy in view of Emer has taught a method as described in claim 21. Joy has further taught allocating available processing time of the processor among at least the first and second threads according to the execution schedule. See column 3, lines 33-51.

38. Referring to claim 23, Joy in view of Emer has taught a method as described in claim 22. Joy has further taught that the allocating comprises dividing the available execution time into a plurality of quanta, each quanta corresponding to a number of instruction cycles for execution of a thread. See column 3, lines 33-51, and note that each thread will execute for N cycles in one scheduling embodiment.

39. Referring to claim 24, Joy in view of Emer has taught a method as described in claim 23. Joy has further taught that at least one quanta corresponds to a thread that is scheduled to execute periodically after a fixed number of execution cycles. See column 3, lines 33-36.

40. Claims 5-12, 18, and 25-28 are rejected under 35 U.S.C. 103(a) as being unpatentable over Joy in view of Emer and further in view of Ramakrishnan et al., U.S Patent No. 6,085,215 (as applied in the previous Office Action and herein referred to as Ramakrishnan).

41. Referring to claim 5, Joy in view of Emer has taught a system as described in claim 17. Joy in view of Emer has not taught that said hardware thread scheduler includes a thread identifier for identifying at least one hard-real-time (HRT) thread and at least one non-real-time thread and a HRT scheduler for regularly scheduling said HRT thread in available time quanta such that said HRT thread is scheduled to ensure the execution of the HRT in a predetermined time. However, Ramakrishnan has taught such a concept. See column 4, line 52, to column 5, line 3, and the abstract. Note that real-time and general (non real-time) threads are determined and that a real time thread is scheduled during the available time quanta such that it executes in predetermined time. In such a system, real-time threads, which perform time-critical tasks, are given priority over general threads. This is clear because the general threads execute for a minimum time and then after that, if a real-time thread needs processing, then it will preempt that general thread. Furthermore, Ramakrishnan has taught that such a system prevents starvation of threads (since all threads get some time to process) and offers a greater degree of fairness in allocating processing resources to various tasks. See column 4, lines 11-14. As a result, in order to execute time-critical tasks in a non-starving and fair way, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy to be used in a time-critical environment and to include at least one HRT thread and an HRT scheduler, as taught by Ramakrishnan.

42. Referring to claim 6, Joy in view of Emer and further in view of Ramakrishnan has taught a system as described in claim 5. Ramakrishnan has further taught that said time quanta is at least one instruction cycle. See the abstract and note that real-time threads are scheduled for a preselected maximum amount of time. This time is inherently at least one cycle because if it were any less (zero cycles), then the thread would never execute.

43. Referring to claim 7, Joy in view of Emer and further in view of Ramakrishnan has taught a system as described in claim 5. Ramakrishnan has further taught that said hardware thread scheduler schedules a non-real-time (NRT) thread to replace a scheduled HRT thread if said HRT is idle. See Fig.3, step 74, and note that if a real-time thread is idle (has no work), then the schedule is free to replace it with another thread, where another thread is either an NRT or another HRT.

44. Referring to claim 8, Joy in view of Emer and further in view of Ramakrishnan has taught a system as described in claim 5. Ramakrishnan has further taught that said thread scheduler schedules the execution of non-real-time (NRT) threads in quanta not allocated to HRT threads. See the Fig.2A and note that a real-time thread is allocated time and that time is the real-time thread's time (step 52). After the HRT is done, an NRT may be scheduled. That is, they are both not scheduled in the same quanta (at the same time). The HRT is to execute (assuming the HRT is first to execute), and then the NRT is to execute.

45. Referring to claim 9, Joy in view of Emer and further in view of Ramakrishnan has taught a system as described in claim 8. Ramakrishnan has further taught that said thread scheduler regularly schedules NRT threads to be executed. See the abstract and note that there

may be a plurality of NRTs for scheduling. They are scheduled for minimum times throughout the entire execution process.

46. Referring to claim 10, Joy in view of Emer and further in view of Ramakrishnan has taught a system as described in claim 5. Joy has further taught:

a) a first storage device for storing program instructions, said processor fetching instructions from the first storage device within a first fetch period. See Fig.3, component 330. The instruction cache (I$) will be fetched from during the time that the instructions needed are in the cache.

b) a second storage device for storing program instructions, said processor fetching instructions from the second storage device within a second fetch period. See column 9, line 66. The main memory will be fetched from during the time that the instructions needed are not in the cache.

c) wherein said first fetch period is substantially shorter than said second fetch period. Fetching from a cache is shorter than fetching from main memory, as is known in the art.

47. Referring to claim 11, Joy in view of Emer and further in view of Ramakrishnan has taught a system as described in claim 10. Joy in view of Emer and further in view of Ramakrishnan has not taught that said first storage device for storing program instructions comprises a static RAM. However, Official Notice is taken that virtually all caches are implemented with static RAM (SRAM) and that SRAM and its advantages are well known and accepted in the art. SRAM is fast, which makes it suitable for caches, and unlike DRAM, it does not need to be refreshed in order to maintain its contents. Consequently, for speed and storage ability, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy's instruction cache such that it is implemented in SRAM.

48. Referring to claim 12, Joy in view of Emer and further in view of Ramakrishnan has taught a system as described in claim 10. Joy in view of Emer and further in view of Ramakrishnan has not taught that said second storage device for storing program instructions comprises a flash memory. However, Official Notice is taken that flash-based main memories and their advantages are well known and accepted in the art. A computer hierarchy based upon volatile main memory loses all information in main memory when power is turned off. A flash-based non-volatile main memory, however, reduces or eliminates the lengthy process of obtaining information from disk when power is turned on. Therefore flash main memory based computer system has higher system performance when a program is initially executed than would a volatile main memory based computer system. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy's main memory such that it is a flash-based main memory.

49. Referring to claim 18, Joy in view of Emer and further in view of Ramakrishnan has taught a system as described in claim 5. Emer has further taught that said time quanta is exactly one instruction cycle. See Fig.1(b), and column 1, lines 52-65.

50. Referring to claim 25, Joy in view of Emer has taught a method as described in claim 21. Joy in view of Emer has not taught identifying at least one hard real-time (HRT) thread and at least one non real-time (NRT) thread. However, Ramakrishnan has taught the concept of real-time threads and general (non-real-time). See column 4, line 52, to column 5, line 3, and the abstract. Note that real-time threads are identified and are those that perform time-critical work and therefore should be given priority in the system over general threads. This priority concept is clear in Ramakrishnan because the general threads execute for a minimum time and then after

that, if a real time thread needs processing, then it will preempt that general thread. Furthermore, Ramakrishnan has taught that such a system gets important work done while preventing starvation of threads (since all threads get some time to process) and offering a greater degree of fairness in allocating processing resources to various tasks. See column 4, lines 11-14. As a result, in order to execute time-critical tasks in a non-starving and fair way, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy to identify at least one HRT and at least one NRT thread, as taught by Ramakrishnan.

51. Referring to claim 26, Joy in view of Emer and further in view of Ramakrishnan has taught a method as described in claim 25. Ramakrishnan has further taught scheduling the HRT thread in available time quanta such that said HRT thread is scheduled to ensure the execution of the HRT thread within a predetermined time. See column 4, line 52, to column 5, line 3, and the abstract. Note that HRT threads are given a maximum time in which to execute. This ensures the execution of the HRT within that maximum time.

52. Referring to claim 27, Joy in view of Emer and further in view of Ramakrishnan has taught a method as described in claim 25. Ramakrishnan has further taught scheduling an NRT thread for a quantum allocated for an HRT thread if said HRT thread is idle. See Fig.3, step 74, and note that if a real-time thread is idle (has no work), then the schedule is free to replace it with another thread, where another thread is either an NRT or another HRT.

53. Referring to claim 28, Joy in view of Emer and further in view of Ramakrishnan has taught a method as described in claim 25. Ramakrishnan has further taught scheduling NRT threads in quanta not allocated for HRT threads. See the Fig.2A and note that a real-time thread is allocated time and that time is the real-time thread's time (step 52). After the HRT is done, an

NRT may be scheduled. That is, they are both not scheduled in the same quanta (at the same time). The HRT is to execute (assuming the HRT is first to execute), and then the NRT is to execute.

54. Claim 14 is rejected under 35 U.S.C. 103(a) as being unpatentable over Joy in view of Emer and further in view of Borkenhagen.

55. Referring to claim 14, Joy in view of Emer has taught a system as described in claim 17. Joy in view of Emer has not explicitly taught that said processor is capable of storing said second thread state of said processor during execution of said first program thread. However, Borkenhagen has taught that a thread switch control register may be implemented for each thread for holding a state of that thread and that the control state for the second thread may be stored during execution of the first thread. See column 13, lines 20-45. This control register (and control state) allows the system to specify which types of events would result in the switching of the associated thread, thereby increasing flexibility by allowing the user to choose how the thread may or may not be switched. Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy in view of Emer to include the control register of Borkenhagen as part of the thread state, where the thread state of the second thread is stored during execution of the first thread.

56. Claim 15 is rejected under 35 U.S.C. 103(a) as being unpatentable over Joy in view of Emer and further in view of Levy et al., U.S. Patent No. 6,314,511 (herein referred to as Levy).

57. Referring to claim 15, Joy in view of Emer has taught a system as described in claim 17. While Joy has taught that said first set of data storage devices comprises registers (see column 8, lines 27-44, and column 3, lines 3-10), Joy has not taught that the registers are shared by a plurality of threads. Instead, Joy has taught that each thread gets its own register file. However, Levy has taught that some tests have shown that shared registers provide performance gains when compared to dedicated per-thread register designs, as taught by Joy. In addition, by sharing registers, the total size of the register file is reduced (since you don't have to have a separate register file for each thread) without sacrificing performance. See Levy, column 8, line 65, to column 9, line 3. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy such that the registers are shared by the threads instead of replicated.

58. Claims 34-41 and 48-55 are rejected under 35 U.S.C. 103(a) as being unpatentable over Borkenhagen in view of Ramakrishnan.

59. Referring to claim 34, Borkenhagen has taught a system as described in claim 1. Borkenhagen has not taught that said hardware thread scheduler includes a thread identifier for identifying at least one hard-real-time (HRT) thread and at least one non-real-time thread and a HRT scheduler for regularly scheduling said HRT thread in available time quanta such that said HRT thread is scheduled to ensure the execution of the HRT in a predetermined time. However, Ramakrishnan has taught such a concept. See column 4, line 52, to column 5, line 3, and the abstract. Note that real-time and general (non real-time) threads are determined and that a real time thread is scheduled during the available time quanta such that it executes in predetermined

time (i.e., in a preselected maximum time). In such a system, real-time threads, which perform time-critical tasks, are given priority over general threads. This is clear because the general threads execute for a minimum time and then after that, if a real time thread needs processing, then it will preempt that general thread. Furthermore, Ramakrishnan has taught that such a system prevents starvation of threads (since all threads get some time to process) and offers a greater degree of fairness in allocating processing resources to various tasks. See column 4, lines 11-14. As a result, in order to execute time-critical tasks in a non-starving and fair way, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Borkenhagen be used in a time-critical environment and to include at least one HRT thread and an HRT scheduler, as taught by Ramakrishnan.

60. Referring to claim 35, Borkenhagen in view of Ramakrishnan has taught a system as described in claim 34. Ramakrishnan has further taught that said time quanta is at least one instruction cycle. See the abstract and note that real-time threads are scheduled for a preselected maximum amount of time. This time is inherently at least one cycle because if it were any less (zero cycles), then the thread would never execute.

61. Referring to claim 36, Borkenhagen in view of Ramakrishnan has taught a system as described in claim 34. Ramakrishnan has further taught that said hardware thread scheduler schedules a non-real-time (NRT) thread to replace a scheduled HRT thread if said HRT is idle. See Fig.3, step 74, and note that if a real-time thread is idle (has no work), then the schedule is free to replace it with another thread, where another thread is either an NRT or another HRT.

62. Referring to claim 37, Borkenhagen in view of Ramakrishnan has taught a system as described in claim 34. Ramakrishnan has further taught that said hardware thread scheduler

schedules the execution of non-real-time (NRT) threads in quanta not allocated to HRT threads. See the Fig.2A and note that a real-time thread is allocated time and that time is the real-time thread's time (step 52). After the HRT is done, an NRT may be scheduled. That is, they are both not scheduled in the same quanta (at the same time). The HRT is to execute (assuming the HRT is first to execute), and then the NRT is to execute.

63. Referring to claim 38, Borkenhagen in view of Ramakrishnan has taught a system as described in claim 37. Ramakrishnan has further taught that said hardware thread scheduler regularly schedules NRT threads to be executed. See the abstract and note that there may be a plurality of NRTs for scheduling.

64. Referring to claim 39, Borkenhagen in view of Ramakrishnan has taught a system as described in claim 34. Borkenhagen has further taught:

a) a first storage device for storing program instructions, said processor fetching instructions from the first storage device within a first fetch period. See Fig.1, component 150. The cache will be fetched from during the time that the instructions needed are in the cache.

b) a second storage device for storing program instructions, said processor fetching instructions from the second storage device within a second fetch period. See Fig.1, component 140. The main memory will be fetched from during the time that the instructions needed are not in the cache.

c) wherein said first fetch period is substantially shorter than said second fetch period. See column 3, lines 13-17. Fetching from a cache is shorter than fetching from main memory.

65. Referring to claim 40, Borkenhagen in view of Ramakrishnan has taught a system as described in claim 39. Borkenhagen has not taught that said first storage device for storing

program instructions comprises a static RAM. However, Official Notice is taken that virtually all caches are implemented with static RAM (SRAM) and that SRAM and its advantages are well known and accepted in the art. SRAM is fast, which makes it suitable for caches, and unlike DRAM, it does not need to be refreshed in order to maintain its contents. Consequently, for speed and storage ability, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Borkenhagen's instruction cache such that it is implemented in SRAM.

66. Referring to claim 41, Borkenhagen in view of Ramakrishnan has taught a system as described in claim 39. Borkenhagen has not taught that said second storage device for storing program instructions comprises a flash memory. However, Official Notice is taken that flash-based main memories and their advantages are well known and accepted in the art. A computer hierarchy based upon volatile main memory loses all information in main memory when power is turned off. A flash-based non-volatile main memory, however, reduces or eliminates the lengthy process of obtaining information from disk when power is turned on. Therefore flash main memory based computer system has higher system performance when a program is initially executed than would a volatile main memory based computer system. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Borkenhagen's main memory such that it is a flash-based main memory.

67. Referring to claim 48, Borkenhagen has taught a method as described in claim 46. Borkenhagen has not taught identifying which of the said program threads said processor executes according to a hard real-time (HRT) execution schedule. However, Ramakrishnan has taught the concept of real-time threads. See column 4, line 52, to column 5, line 3, and the

abstract. Note that real-time threads are identified and are those that perform time-critical work and therefore should be given priority in the system. In such a system, real-time threads are given priority over general threads (note that Borkenhagen has also taught using priority with threads in the abstract). This priority concept is clear in Ramakrishnan because the general threads execute for a minimum time and then after that, if a real time thread needs processing, then it will preempt that general thread. Furthermore, Ramakrishnan has taught that such a system prevents starvation of threads (since all threads get some time to process) and offers a greater degree of fairness in allocating processing resources to various tasks, while allowing important work to get done. See column 4, lines 11-14. As a result, in order to execute time-critical tasks in a non-starving and fair way, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Borkenhagen to be used in a time-critical environment and to identify which of the program threads the processor executes according to an HRT execution schedule, as taught by Ramakrishnan.

68. Referring to claim 49, Borkenhagen in view of Ramakrishnan has taught a method as described in claim 48. Borkenhagen has further taught allocating available processing time of the processor among at least the first and second threads according to the predetermined fixed execution schedule. See claim 5 of Borkenhagen and recall that a first thread is executed for a certain amount of time and then a second thread is executed for a certain amount of time. Given only two threads, this cycle will repeat.

69. Referring to claim 50, Borkenhagen in view of Ramakrishnan has taught a method as described in claim 49. Borkenhagen has further taught that the allocating comprises dividing the available execution time into a plurality of quanta, each quanta corresponding to a number of

instruction cycles for execution of a thread. Again, see claim 5 of Borkenhagen. Each thread is allocated a number of cycles (quanta) in which to execute. This number is associated with the timeout value. The overall processing time will be divided into an amount of time for the first thread and an amount of time for the second thread.

70. Referring to claim 51, Borkenhagen in view of Ramakrishnan has taught a method as described in claim 50. Borkenhagen has further taught that at least one quantum corresponds to a thread that is scheduled to execute periodically after a fixed number of execution cycles. A thread in Borkenhagen will execute after a fixed number of cycles in which another thread executes. For instance, thread A might execute for 10 cycles, and thread B might execute for 5 cycles. Five cycles into thread B's execution, thread A will be switched in. This happens every time the system gets 5 cycles into thread B's execution (it's fixed).

71. Referring to claim 52, Borkenhagen in view of Ramakrishnan has taught a method as described in claim 48. Ramakrishnan has further taught that identifying further comprises identifying at least one hard real-time (HRT) thread and at least one non real-time (NRT) thread. See column 4, line 52, to column 5, line 3, and the abstract. Note that real-time threads are identified and are those that perform time-critical work and therefore should be given priority in the system over general threads, which are the less time-critical threads.

72. Referring to claim 53, Borkenhagen in view of Ramakrishnan has taught a method as described in claim 52. Ramakrishnan has further taught scheduling the HRT thread in available time quanta such that said HRT thread is scheduled to ensure the execution of the HRT thread within a predetermined time. See column 4, line 52, to column 5, line 3, and the abstract. Note

that HRT threads are given a maximum time in which to execute. This ensures the execution of the HRT within that maximum time.

73. Referring to claim 54, Borkenhagen in view of Ramakrishnan has taught a method as described in claim 52. Ramakrishnan has further taught scheduling an NRT thread for a quantum allocated for an HRT thread if said HRT thread is idle. See Fig.3, step 74, and note that if a real-time thread is idle (has no work), then the schedule is free to replace it with another thread, where another thread is either an NRT or another HRT.

74. Referring to claim 55, Borkenhagen in view of Ramakrishnan has taught a method as described in claim 52. Ramakrishnan has further taught scheduling NRT threads in quanta not allocated for HRT threads. See the Fig.2A and note that a real-time thread is allocated time and that time is the real-time thread's time (step 52). After the HRT is done, an NRT may be scheduled. That is, they are both not scheduled in the same quanta (at the same time). The HRT is to execute (assuming the HRT is first to execute), and then the NRT is to execute.

75. Claim 44 is rejected under 35 U.S.C. 103(a) as being unpatentable over Borkenhagen in view of Levy.

76. Referring to claim 44, Borkenhagen has taught a system as described in claim 1. While Borkenhagen has hinted at sharing of resources (column 5, lines 56-57), Borkenhagen has not explicitly taught that said first set of storage devices comprises registers shared by a plurality of threads. However, Levy has taught that some tests have shown that shared registers provide performance gains when compared to dedicated per-thread register designs. In addition, by sharing registers, the total size of the register file is reduced (since you don't have to have a

separate register file for each thread) without sacrificing performance. See Levy, column 8, line 65, to column 9, line 3. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Borkenhagen such that the registers are shared by the threads.

Response to Arguments

77. Regarding applicant's arguments, on pages 15-17 of the remarks filed on March 27, 2007, with respect to the 35 U.S.C 112, 2nd paragraph, rejection, applicant's attention is directed to the 112 rejection set forth above for a more detailed explanation as to why the rejection is being maintained. In addition, a couple references have been provided to applicant for showing the concept behind a clock cycle, i.e., an instruction cycle, and how there is no between cycles. As long as the clock is running, the system is in a cycle. Both the rising and falling edges are part of the cycle. Note that even though the dates of the documents are after applicant's effective filing date, the subject matter is well known and has not changed over the years. However, these documents explain the concept well, and that is why they are being cited.

78. Applicant argues the novelty/rejection of claims 1 and 46 on pages 18-21 of the remarks, in substance that:

> "The time-out register in Borkenhagen is used "to force a thread switch to the dormant thread after some time if no useful processing is being accomplished to prevent the system from hanging." Borkenhagen, col. 14, lines 49-51. This forced thread switch prevents a thread from "spinning in a loop unable to do useful work" because it is unable to acquire ownership of, or access to, a necessary resource. Borkenhagen, col. 14, lines 31-34. Thus, the time-out register in Borkenhagen does not specify the processing time allocated to the first and second threads, but rather specifies a maximum time the first and second threads can be inactive before forcing a thread switch."

79. These arguments are not found persuasive for the following reasons:

a) While applicant is correct in stating that the time-out register specifies inactivity time before switching, this is just one feature of the register. Column 14, lines 40-51, discussing allocating a maximum amount of processing time for a thread that might seldom experience long-latency events (i.e., for an active thread) and for threads that are inactive (supported by the "Yet another excessive delay..." language). Furthermore, column 15, lines 1-7 explain that when a thread switch occurs, the new timeout value is decremented **each cycle until the decrement register value equals zero** and not *decremented each cycle the thread is inactive until the value equals zero*, as applicant suggests. That is, if a thread has an associated timeout of 100 cycles, that thread will run for a maximum of 100 cycles whether it is active or inactive.

80. Applicant argues the novelty/rejection of claim 17 on page 23 of the remarks, in substance that:

> "Emer simply does not address the time required to switch between threads. Moreover, Emer explicitly discloses: 'On any given cycle, a processor executes instructions from just one of the threads. On the next cycle, it switches to a different thread context and executes instructions from the new thread.'"

81. These arguments are not found persuasive for the following reasons:

a) Emer clearly shows in Fig.1(b) what happens when threads are switched every N cycles, where N = 1. Specifically, a fine-grained multi-threaded system is achieved where a new thread executes every cycle. Fine-grained systems are well known and accepted in the art. Note that there is no time penalty because the system is able to perform the maximum amount of work per cycle, i.e., execute one instruction from a given thread. If a time penalty were incurred, then there would have to be some idle time in the processor which is not seen in Fig.1(b). Since there is no idle time, there is no time penalty. In turn, since there is no time penalty, Emer switches

between instructions/cycles, as applicant has equated the two in applicant's specification on page 17, lines 9-13.

82. Applicant argues the novelty/rejection of claim 17 on page 24 of the remarks, in substance that:

> "Moreover, the combination of Joy and Emer does not disclose either the "thread selection hardware in the pipelined processor [switching] from said first thread state to said second thread state between consecutive instruction cycles" of claim 17 or the "switching...between the end of an execution cycle and before the beginning of a next consecutive instruction cycle" of claim 19. By combining the "at least one cycle" thread switch delay described in Joy, with the "effective" elimination of vertical waste of Emer, the teachings of the combined references amount to no more than what Joy discloses, that is, thread switching that requires an overhead of at least one cycle. Thus, the claimed thread switching "between consecutive instruction cycles" or "between the end of an execution cycle and before the beginning of a next consecutive execution cycle" is not disclosed by the cited references, both alone and in combination.
> "

83. These arguments are not found persuasive for the following reasons:

a) As described above, the idea extracted from Emer is merely that threads may be switched every cycle without time penalty, which is clear from Fig.1(b). Whatever logic allows threads to be switched every cycle in Emer would be implemented in Joy in order to achieve a thread switch every cycle. Clearly, Joy's logic would be replaced because Joy's logic causes overhead, as pointed out in applicant's argument, while Emer's logic eliminates vertical waste, which is advantageous for throughput (this is the motivation for combination).

84. The arguments on the remaining pages include explaining how additional cited references do not cure the deficiencies of Joy and/or Emer and/or Borkenhagen. However, as described above, these references do not have deficiencies.

Conclusion

85. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to David J. Huisman whose telephone number is (571) 272-4168. The examiner can normally be reached on Monday-Friday (8:00-4:30).

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (571) 272-4162. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

RICHARD L. ELLIS
PRIMARY EXAMINER

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

DJH
David J. Huisman
May 17, 2007